09/480,309

[40101/09301]

AF
IFW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s)	:	Wilner
Serial No.	:	~~09/840,309~~ 8
Filing Date	:	January 10, 2000
For	:	Protection Domains for a Computer Operating System
Group Art Unit	:	2195
Examiner	:	Syed J. Ali

Mail Stop: Appeal Brief-Patent
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Certificate of Mailing

I hereby certify that this correspondence is being deposited with U.S. Postal Services as first class mail in an envelope addressed to:

Mail Stop: Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

By: _____ Date: February 28, 2006
Michael J. Marcin, (Reg. No. 48,198)

TRANSMITTAL

In response to the Notice of Appeal filed November 29, 2005 and the Advisory Action dated December 29, 2005, transmitted herewith please find an Appeal Brief (in triplicate) for filing in the above-identified application. Applicants request a one-month extension. Please charge the Credit Card of **Fay Kaplun & Marcin, LLP** in the amount of $620.00 (PTO-Form 2038 is enclosed herewith). The Commissioner is hereby authorized to charge the **Deposit Account of Fay Kaplun & Marcin, LLP NO. 50-1492** for any additional required fees. A copy of this paper is enclosed for that purpose.

Respectfully submitted,

Dated: February 28, 2006

By: _____
Michael J. Marcin, Reg. 48,198

03/07/2006 MAHMED1 00000002 09480309
02 FC:1251 120.00 OP

Fay Kaplun & Marcin, LLP
150 Broadway, Suite 702
New York, NY 10038
Tel: (212) 619-6000
Fax: (212) 619-0276

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:	)	
	)	
David N. Wilner	)	
	)	
Serial No.: 09/480,309	)	Group Art Unit: 2195
	)	
Filed: January 10, 2000	)	Examiner: Syed J. Ali
	)	
For: PROTECTION DOMAINS FOR A COMPUTER OPERATING SYSTEM	)	**Board of Patent Appeals and Interferences**

Mail Stop: Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF UNDER 37 C.F.R. § 41.37

In support of the Notice of Appeal filed November 29, 2005, and pursuant to 37 C.F.R. § 41.37, Appellant presents their appeal brief in the above-captioned application.

This is an appeal to the Board of Patent Appeals and Interferences from the Examiner's final rejection of claims 1-3, 5, 8-10, 13-23, and 25-27 in the final Office Action dated August 29, 2005. The appealed claims are set forth in the attached Claims Appendix.

1. Real Party in Interest

This application is assigned to Wind River Systems, the real party in interest.

2. Related Appeals and Interferences

There are no other appeals or interferences which would directly affect, be directly affected, or have a bearing on the instant appeal.

3. Status of the Claims

Claims 1-3, 5, 8-10, 13-23, and 25-27 have been rejected in the final Office Action. The final rejection of claims 1-3, 5, 8-10, 13-23, and 25-27 is being appealed.

4. Status of Amendments

All amendments submitted by the appellant have been entered. None were submitted after the Advisory Action.

5. Summary of Claimed Subject Matter

The present invention relates to a system and method for implementing a "protection domain" system. Specifically, the present invention describes a method that loads a code module into a memory space of a first domain that owns one of a kernel space and a portion of a user space. (See *Specification*, p. 17, ll. 19-22). The code module will include an instruction that has a symbol reference. The location of that symbol reference is determined whether it is within the memory space. (See *Specification*, p. 17, l. 22 – p. 18, l. 4). If it is determined that the symbol reference is in an external location, a link stub is generated for that

symbol reference (See *Specification*, p. 18, ll. 18-21) and the instruction to the link stub is redirected (See *Specification*, p. 26, ll. 15-16). A determination is made if the external location is within a second domain owning one of a kernel space and a portion of a user space that is within a protection view of the first domain. (See *Specification*, p. 29, ll. 18-20). If the second domain is determined to not be within the protection view of the first domain, a request to attach the second domain to the first domain is made (See *Specification*, p. 29, l. 22 – p. 30, l. 1) and then the second domain is attached using an attachment mechanism (See *Specification*, p. 22, ll. 18-21).

The present invention also describes a method that creates a task that executes instructions in a first domain that owns one of a kernel space and a portion of a user space. (See *Specification*, p. 33, ll. 19-24). A first jump instruction is executed in the instructions that refer to a link stub corresponding to an external location in a second domain that also owns one of a kernel space and a potion of a user space. (See *Specification*, p. 34, ll. 4-7). Upon execution of the link stub (See *Specification*, p. 34, ll. 11-14), the external location is compared to a task protection view (See *Specification*, p. 35, ll. 10-17). If the external location is outside the task protection view, a processing exception is generated (See *Specification*, p. 36, ll. 3-5) and an exception handling routine is executed (See *Specification*, p. 36, ll. 5-9). The exception handling routine includes saving a pre-exception setting of the task protection view, altering the task protection view to include a protection view of the second domain, and jumping to the external location. (See *Specification*, p. 36, ll. 11-17).

The present invention also describes a computer system for implementing a "protection domain" system. The computer system has memory locations contained in a system space that includes a kernel space and at least one user space. (See *Specification*, p. 13, l. 22 – p.

14, l. 9). The computer system also has protection domains where at least one protection domain owns a portion of the system space and each protection domain includes a protection view that defines unprotected access for a set of the number of protection domains. (See *Specification*, p. 14, ll. 18-22, and p. 16, ll. 4-16).

6. Grounds of Rejection to be Reviewed on Appeal

I. Whether claims 1-3, 5, 8-10, 13-23, and 25-27 are unpatentable under 35 U.S.C. § 103(a) as obvious over U.S. Patent No. 5,359,721 to Kempf et al. ("the Kempf patent") in view of U.S. Patent No. 4,430,705 to Cannavino et al. (the "Cannavino" patent) in view of U.S. Patent No. 6,256,657 to Chu ("the Chu patent").

7. Grouping of Claims

Claims 1-3, 5, 8-10, 13-23, and 25-27 may stand or fall together.

8. Argument

I. The Rejection of Claims 1-3, 5, 8-10, 13-23, and 25-27 Under 35 U.S.C. § 103(a) as Being Obvious Over U.S. Patent No. 5,359,721 to Kempf et al. in View of U.S. Patent No. 4,430,705 to Cannavino et al. in View of U.S. Patent No. 6,256,657 to Chu Should Be Reversed.

A. The Examiner's Rejection

In the final Office Action, the Examiner rejected claims 1-3, 5, 8-10, 13-23, and 25-27 under 35 U.S.C. § 103(a) as being unpatentable over the Kempf patent in view of the Cannavino patent in view of the Chu patent. (See 08/29/05 *Office Action*, p. 2, ll. 9-11).

The Kempf patent describes a method of allowing a process executing in non-

supervisor mode to perform dynamic linking across address spaces without compromising system security. (See Kempf, col. 2, ll. 50-52). The Kempf patent compares this to a typical process executing in non-supervisor mode, which cannot access another process executing in another address space without invoking the operating system. (See Kempf, col. 1, ll. 24-27). Although a process can dynamically link shared libraries into itself when it starts up without entering supervisor mode, security may be compromised. (See Kempf, col. 2, ll. 23-29).

The Cannavino patent describes an authorization mechanism for establishing addressability to information in another address space. This mechanism permits a program in one address space to access data in another address space without invoking a supervisor. (See Cannavino, Abstract). A subsystem control facility provides basic authority control with dual address space memory references, program subsystem linkages, and Address Space Number ("ASN") translation to main memory addresses with authorization control. (See Cannavino, col. 3, ll. 9-14). Basic authority control makes a level privilege or authority available to problem programs. (See Cannavino, col. 3, ll. 15-16). The dual address space concept provides the problem program with the ability to move information from one address space into another, and also to specify in which address space the operands of the program are to be accessed. (See Cannavino, col. 3, ll. 53-57). This incorporates a segment table, which is used to translate virtual addresses of a particular address space. (See Cannavino, col. 3, ll. 57-64). The subsystem linkage control provides for direct linkage between problem programs by authorizing the execution of a Program Call and Program Transfer instruction (See Cannavino, col. 4, ll. 17-24). The ASN feature provides translation tables and authorization controls whereby a program in the problem state can designate an address space as being a primary address space or a secondary address space. (See Cannavino, col. 4, l. 66 – col. 5, l. 2).

The Chu patent describes a method for transferring data held within the kernel space to the user space by remapping the virtual memory areas of the kernel and user spaces via the use of an intermediate memory area. (See Chu, col. 6, ll. 21-34). As the Examiner points out, this is a well-known principle for protection domains, and conventional virtual memory address remapping supports such attachment. (See 8/29/05 Office Action, p. 4, ¶ 7).

B. The Cited Patents Do Not Disclose When the System Space Comprises a Number of Memory Locations And a Number of Protection Domains Wherein the First Domain Owns One of a Kernel Space And a Portion of a User Space Comprising a Memory Space And a Protection View Designating a Set of Protection Domains For Unrestricted Memory Access, and as Recited in Claim 1.

The Examiner asserts that the Kempf patent discusses the traditional operating system approach of switching into supervisor mode before discussing a work-around that allows for non-supervisor mode cross address space linking. (See Advisory Action, p. 2, ll. 6-7). Since the Kempf patent addresses processes executing in non-supervisor mode, the Examiner's claim is unfounded.

As discussed above in the Summary of Claimed Subject Matter, the code module is loaded into a memory space of a first domain, wherein the first domain owns one of a kernel and a portion of a user space. A process requiring a user application to switch into a kernel space falls into a category where switching into supervisor mode is required. On the other hand, the Kempf patent, more generally, is when a program code manager provides an object oriented interface to a client process, thereby facilitating access to a program code segment object comprising executable binary code of a program. (See Kempf, col. 6, ll. 42-46). Thus, the Kempf patent does not fall into the same category where a user application interacts with a kernel space (*i.e.*, switching into supervisor mode). More specifically, the Kempf patent falls

into a category where one user application interacts with another user application, where the processes are executed in non-supervisor mode. Accordingly, claim 1 differs significantly from the teachings of Kempf, since the operations of each respective process occur within differing memory locations.

The Examiner argues that such an approach of switching into a supervisor mode is not necessarily required. (See Advisory Action, p. 2, ll. 4-5). However, the language in the cited reference states that "traditional operating systems typically require a process to enter supervisor mode before it can dynamically load a program into an address space." (See Kempf, col. 1, ll. 36-39). Furthermore, the Applicants' invention deals with a kernel space which is a process executed in supervisor mode. Additionally, the Kempf patent discusses the work-around for processes running in non-supervisor mode without compromising the entire system. (See Kempf, col. 2, ll. 37-44). Thus, the Kempf patent discusses subject matter that is wholly different from claim 1.

Thus, it is respectfully submitted that the Kempf patent does not teach or disclose "loading a code module into a memory space of a first domain, wherein the first domain owns one of a kernel space and a portion of a user space, the code module including an instruction having a symbol reference" as also cited in claim 1.

In addition, the Examiner acknowledged that the Kempf patent fails to teach or suggest a method for handling "several potential irregularities including exception handling for unauthorized accesses and attachment of separate address spaces." (See 8/29/05 Office Action, p. 3, ¶ 7). However, the Examiner further stated that the Cannavino patent shows these claim elements, thereby rendering the claimed subject matter obvious over the Kempf patent. (See Id.). Appellant respectfully disagrees with the Examiner's rejection of claim 1.

It is respectfully submitted that the Cannavino patent neither discloses nor suggests "determining if the external location is within a second domain that is within a protection view of the first domain, wherein the second domain owns the other one of the kernel space and a portion of the user space," as recited in claim 1. In Cannavino, the Examiner points to a disclosure of a method by which a problem program executing in a first address space obtains access to data in a second address space by executing a Set Second ASN ("SSAR") instruction. (See 8/29/05 Office Action, p. 3, ¶ 5). In this method, each address space has an associated set of address translation tables. (See Cannavino, col. 15, ll. 66-68). These tables are used to locate address space control parameters, as well as third authority table used to perform authorization tests (See Cannavino, col. 11, ll. 48-54). Primary and secondary table descriptors, which point to the first and second address spaces respectively, are stored in their respective control registers (See Cannavino, col. 15, l. 62 – col. 16, l. 3). As the program executing in the first address space attempts to access data in the second address space, the descriptor pointing to the second address space is stored into the secondary control register. (See Cannavino, col. 15, l. 68 – col. 16, l. 3). The address in the secondary control register may then be compared to the address in the primary control register to determine if an address translation operation must be performed to obtain data in the other address space. (See Cannavino, col. 16, ll. 3-10). Thus, the Cannavino patent only requires a determination that the object being referenced is not within a set of domains.

In the present invention, each protection domain of the present invention includes a mechanism allowing tasks executing in one protection domain to access resources and objects in a separate protection domain. (See Application, p. 11, ll. 5-7). One such mechanism is a protection view, included in each protection domain, which defines system resources and objects

to which it has access. (See Application, p. 11, ll. 7-9). In other words, the objects that are accessible by the domain are specified, and thus, not merely a determination that the object being referenced is not within a protection view. Upon creation of protection domains, a set of protection domain attributes that may be used to control the actions allowed by tasks executing in a protection domain are specified. (See Application, p. 17, ll. 4-6). A second domain may be found in the protection view of a first domain if the first domain has been attached to the second domain. (See Application, p. 11, ll. 11-14). The Examiner acknowledges that neither Kempf nor Cannavino reference a protection view. (See 8/29/05 Office Action, p. 3, ¶ 7). Accordingly, the Examiner incorrectly equates the protection view of the present invention with the address space tables or ASNs described in Cannavino. (Id.). Thus, the present invention goes beyond merely requiring a determination that the object being referenced is not within a protection view, but specifies the objects that are accessible by the domain.

It is respectfully submitted that neither the Kempf patent nor the Cannavino patent, either alone or in combination, disclose or suggest "loading a code module into a memory space of a first domain, wherein the first domain owns one of a kernel space and a portion of a user space, the code module including an instruction having a symbol reference" and "determining if the external location is within a second domain that is within a protection view of the first domain, wherein the second domain owns the other one of the kernel space and a portion of the user space" as recited in claim 1. The Chu patent does not cure the defects of Kempf or Cannavino. Accordingly, because attaching the second domain to the first domain using an attachment mechanism, as recited in claim 1, is applied to a situation where a user process must enter supervisor mode and the protection view must specify the objects that are accessible by the domain, it is respectfully submitted that claim 1 is allowable over the cited references. Because

claims 2, 3, and 5 depend from, and therefore include all the limitations of claim 1, it is respectfully submitted that these claims are also allowable.

Claim 8 includes substantially the same limitations as claim 1, including "creating a task in a first domain, wherein the first domain owns one of a kernel space and a portion of a user space, the task executing a number of instructions" and "executing a first jump instruction in the number of instructions that refers to a link stub corresponding to an external location in a second domain, wherein the second domain owns the other one of the kernel space and a portion of the user space." Thus, for the reasons described above with reference to claim 1, it is respectfully submitted that claim 8 is also allowable. Because claims 9, 10, 13, and 14 depend from, and therefore include all of the limitations of claim 8, it is respectfully submitted that these claims are also allowable.

Claim 15 includes substantially the same limitations as claim 1, including "a system space having a number of memory locations, wherein the system space includes a kernel space and at least one user space." Thus, for the reasons described above with reference to claim 1, it is respectfully submitted that claim 15 is also allowable. Because claims 16-23 and 25-27 depend from, and, therefore include all of the limitations of claim 15, it is respectfully submitted that these claims are also allowable.

9. Conclusions

For the reasons set forth above, Appellant respectfully requests that the Board reverse the final rejections of the claims by the Examiner under 35 U.S.C. § 103(a), and indicate that claims 1-3, 5, 8-10, 13-23, and 25-27 are allowable.

Respectfully submitted,

Date: February 28, 2006

By: _____
Michael J. Marcin (Reg. No. 48,198)

Fay Kaplun & Marcin, LLP
150 Broadway, Suite 702
New York, NY 10038
Tel: (212) 619-6000
Fax: (212) 619-0276

CLAIMS APPENDIX

1. A method, comprising the steps of:

loading a code module into a memory space of a first domain, wherein the first domain owns one of a kernel space and a portion of a user space, the code module including an instruction having a symbol reference;

determining if the symbol reference is to an external location outside of the memory space;

generating a link stub for the symbol reference when the symbol reference is to an external location to access the external location;

redirecting the instruction to the link stub; and

determining if the external location is within a second domain that is within a protection view of the first domain, wherein the second domain owns the other one of the kernel space and a portion of the user space;

requesting attachment of the second domain to the first domain when the second domain is determined not to be within the protection view of the first domain; and

attaching the second domain to the first domain using an attachment mechanism.

2. The method of claim 1, wherein the link stub is part of a linking table entry corresponding to the symbol reference.

3. The method of claim 1, wherein the link stub is a jump instruction to the external location.

4. Cancelled

5. The method of claim 1, further comprising the steps of:

determining whether the attachment request is permitted based on authorization information provided by the first domain;

wherein attachment to the second domain is not permitted when the attachment request is not permitted.

6. Cancelled

7. Cancelled

8. A method, comprising:

creating a task in a first domain, wherein the first domain owns one of a kernel space and a portion of a user space, the task executing a number of instructions;

executing a first jump instruction in the number of instructions that refers to a link stub corresponding to an external location in a second domain, wherein the second domain owns the other one of the kernel space and a portion of the user space;

executing the link stub;

comparing the external location to a task protection view;

generating a processing exception when the external location is outside the task protection view; and

executing an exception handling routine in response to the generation of the processing exception, the exception handling routine including,

saving a pre-exception setting of the task protection view,

altering the task protection view to include a protection view of the second domain, and

jumping to the external location.

9. The method of claim 8, wherein the link stub is part of linking table entry corresponding to the external location.

10. The method of claim 8, wherein the link stub includes a second jump instruction to the external location.

11. Cancelled

12. Cancelled

13. The method of claim 8, wherein the task protection view is saved on a task protection switch stack.

14. The method of claim 8, further comprising steps of:

retrieving the pre-exception setting of the task protection view;

restoring the task protection view using the pre-exception setting of the task protection view;

returning to a subsequent instruction to the first jump instruction in the number of instructions.

15. A computer system, comprising:

a system space having a number of memory locations, wherein the system space includes a kernel space and at least one user space;

a number of protection domains, at least one of the number of protection domains owning a portion of the system space, wherein each of the number of protection domains includes a protection view defining a set of the number of protection domains to which unprotected access may be made.

16. A computer system of claim 15, wherein the at least one of the number of protection domains includes at least one of

a code module,

a link stub, and

an entry point.

17. The system of claim 16, wherein the link stub is part of a linking table entry in a linking table.

18. The system of claim 16, wherein the entry point is represented in a symbol table.

19. The system of claim 16, wherein at least one of the number of protection domains is a system protection domain that includes:

at least on code module including executable code for operating system services; and

at least one system object owned by the system protection domain.

20. The system of claim 19, wherein the system protection domain includes a protection domain list that includes entries for each of the number of protection domains.

21. The system of claim 19, wherein at least one of the number of protection domains is a first protection domain that includes:

at least one code module including executable code for a first set of functions; and

a number of link stubs;

wherein at least one of the link stubs corresponds to a symbol referenced in the executable code for the first set of functions, and such at least one link stub includes executable code to direct execution to the executable code for operating system services.

22. The system of claim 21, wherein the number of protection domains includes a second protection domain that includes:

at least one code module including executable code for a second set of functions; and

a number of entry points;

wherein each of the entry points corresponds to a symbol in the executable code for the second set of functions.

23. The system of claim 22, wherein one of the link stubs of the first protection domain corresponds to one of the number of entry points in the second protection domain, and such link stub includes executable code to direct execution to the one of the number of entry points in the second protection domain.

24. Cancelled

25. The system of claim 15, wherein the protection view sets a memory range of allowable memory access, and wherein a memory fault is generated when memory access is attempted outside of the memory range.

26. The system of claim 15, wherein the memory range is contiguous.

27. The system of claim 25, further comprising an exception handling routine that is executed on the occurrence of the memory fault, the exception handling routine including a protection switch mechanism.

28–34. Cancelled

EVIDENCE APPENDIX

No evidence has been entered or relied upon in the present appeal.

RELATED PROCEEDING APPENDIX

No decisions have been rendered regarding the present appeal or any proceedings related thereto.